EWA Splatting

Matthias Zwicker, Hanspeter Pfister, *Member*, *IEEE*,
Jeroen van Baar, and Markus Gross, *Member*, *IEEE*

Abstract—In this paper, we present a framework for high quality splatting based on elliptical Gaussian kernels. To avoid aliasing artifacts, we introduce the concept of a resampling filter, combining a reconstruction kernel with a low-pass filter. Because of the similarity to Heckbert's EWA (elliptical weighted average) filter for texture mapping, we call our technique EWA splatting. Our framework allows us to derive EWA splat primitives for volume data and for point-sampled surface data. It provides high image quality without aliasing artifacts or excessive blurring for volume data and, additionally, features anisotropic texture filtering for point-sampled surfaces. It also handles nonspherical volume kernels efficiently; hence, it is suitable for regular, rectilinear, and irregular volume datasets. Moreover, our framework introduces a novel approach to compute the footprint function, facilitating efficient perspective projection of arbitrary elliptical kernels at very little additional cost. Finally, we show that EWA volume reconstruction kernels can be reduced to surface reconstruction kernels. This makes our splat primitive universal in rendering surface and volume data.

Index Terms—Rendering systems, volume rendering, texture mapping, splatting, antialiasing.

1 Introduction

Volume rendering is an important technique in visualizing acquired and simulated datasets in scientific and engineering applications. The ideal volume rendering algorithm reconstructs a continuous function in 3D, transforms this 3D function into screen space, and then evaluates opacity integrals along line-of-sights. In 1989, Westover [1], [2] introduced *splatting* for interactive volume rendering, which approximates this procedure. Splatting algorithms interpret volume data as a set of particles that are absorbing and emitting light. Line integrals are precomputed across each particle separately, resulting in *footprint functions*. Each footprint, or splat, spreads its contribution in the image plane. These contributions are composited back to front into the final image.

On the other hand, laser range and image-based scanning techniques have produced some of the most complex and visually stunning graphics models to date [3], resulting in huge sets of surface point samples. A commonly used approach is generating triangle meshes from the point data and using mesh reduction techniques to render them [4], [5]. In contrast, recent efforts have focused on direct rendering techniques for point samples without connectivity [6], [7], [8]. Most of these approaches are based on a splatting approach similar to splatting in volume rendering.

In this paper, we present a framework for high quality splatting. Our derivation proceeds along similar lines as Heckbert's *elliptical weighted average* (EWA) texture filter [9], therefore, we call our algorithm *EWA splatting*. The main feature of EWA splatting is that it integrates an elliptical Gaussian reconstruction kernel and a low-pass filter, therefore preventing aliasing artifacts in the output image while avoiding excessive blurring. Moreover, we use the same framework to derive splat primitives for volume as well as for surface data.

EWA volume splatting works with arbitrary elliptical Gaussian reconstruction kernels and efficiently supports perspective projection. Our method is based on a novel approach to compute the footprint function, which relies on the transformation of the volume data to so-called *ray space*. This transformation is equivalent to perspective projection. By using its local affine approximation at each voxel, we derive an analytic expression for the EWA footprint in screen space. The EWA volume splat primitive can be easily integrated into conventional volume splatting algorithms. Because of its flexibility, it can be utilized to render rectilinear, curvilinear, or unstructured volume datasets. The rasterization of the footprint is performed using forward differencing, requiring only one 1D footprint table for all reconstruction kernels and any viewing direction.

EWA surface splatting is equivalent to a screen space formulation of the EWA texture filter for triangle rendering pipelines [10]. Hence, it provides high quality, anisotropic texture filtering for point-sampled surfaces. We will show that EWA surface splatting can be derived from EWA volume splatting by reducing Gaussian volume reconstruction kernels to surface reconstruction kernels. Hence, EWA splats are a universal rendering primitive for volume and for surface data. For example, we can perform high quality iso-surface rendering by flattening the 3D Gaussian kernels along the volume gradient.

The paper is organized as follows: We discuss previous work in Section 2. In Section 3, we review fundamental results from signal processing theory that are needed to analyze aliasing. We also present the general concept of an ideal resampling filter that prevents aliasing during rendering by combining a reconstruction kernel and a low-pass

- *M. Zwicker and M. Gross are with the Computer Graphics Laboratory, ETH Zentrum, 8092 Zurich, Switzerland. E-mail: {zwicker, gross}@inf.ethz.ch.*
- *H. Pfister and J. van Baar are with MERL—Mitsubishi Electric Research Laboratories, Cambridge, MA 02139. E-mail: {pfister, jeroen}@merl.com.*

Manuscript received 15 Feb. 2002; revised 15 Mar. 2002; accepted 2 Apr. 2002.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 116207.

1077-2626/02/$17.00 © 2002 IEEE

filter. Next, we describe how to model volume rendering as a resampling process in Section 4, leading to the formulation of an ideal volume resampling filter. Similarly, we derive an ideal resampling filter for rendering point-sampled surfaces in Section 5. In Section 6, we introduce the EWA resampling filter, which uses elliptical Gaussians as reconstruction kernel and as low-pass filter. We present explicit formulas for both the EWA volume resampling filter and the EWA surface resampling filter. Moreover, we show how to derive the surface resampling filter as a special case of the volume resampling filter by flattening the volume reconstruction kernels. Finally, Sections 7 and 8 discuss our implementation and results and Section 9 concludes the paper.

2 Previous Work

The original work on splatting in the context of volume rendering was presented by Westover [1]. Basic volume splatting algorithms suffer from inaccurate visibility determination when compositing the splats from back to front. This leads to visible artifacts, such as color bleeding. Later, Westover [2] solved the problem using an axis-aligned sheet buffer. However, this technique is plagued by disturbing popping artifacts in animations. Recently, Mueller and Crawfis [11] proposed aligning the sheet buffers parallel to the image plane instead of parallel to an axis of the volume data. Additionally, they splat several slices of each reconstruction kernel separately. This technique is similar to *slice-based* volume rendering [12], [13] and does not suffer from popping artifacts. Mueller and Yagel [14] combine splatting with ray casting techniques to accelerate rendering with perspective projection. Laur and Hanrahan [15] describe a hierarchical splatting algorithm enabling progressive refinement during rendering. Furthermore, Lippert and Gross [16] introduced a splatting algorithm that directly uses a wavelet representation of the volume data.

Additional care has to be taken if the 3D kernels are not radially symmetric, as is the case for rectilinear, curvilinear, or irregular grids. In addition, for an arbitrary position in 3D, the contributions from all kernels must sum up to one. Otherwise, artifacts such as splotches occur in the image. For rectilinear grids, Westover [2] proposes using elliptical footprints that are warped back to a circular footprint. To render curvilinear grids, Mao [17] uses stochastic Poisson resampling to generate a set of new points whose kernels are spheres or ellipsoids. He computes the elliptical footprints very similarly to Westover [2]. As pointed out in Section 6.2, our technique can be used with rectilinear, curvilinear, and irregular grids to efficiently and accurately project and rasterize the elliptical splat kernels.

Westover's original framework does not deal with sampling rate changes due to perspective projections. Aliasing artifacts may occur in areas of the volume where the sampling rate of diverging rays falls below the volume grid sampling rate. The aliasing problem in volume splatting has first been addressed by Swan et al. [18] and Mueller et al. [19]. They use a distance-dependent stretch of the footprints to make them act as low-pass filters. In contrast, EWA splatting models both reconstructing and band limiting the texture function in a unified framework.

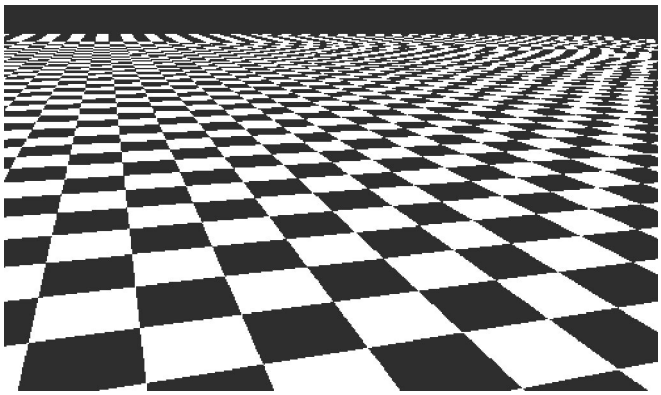


Fig. 1. Aliasing artifacts. Note the moiré patterns and jagged edges.

The concept of representing surfaces as a set of points and using these as rendering primitives has been introduced in a pioneering report by Levoy and Whitted [20]. Due to the continuing increase in geometric complexity, their idea has recently gained more interest. QSplat [6] is a point rendering system that was designed to interactively render large data sets produced by modern scanning devices. Other researchers demonstrated the efficiency of point-based methods for rendering geometrically complex objects [7], [8]. In some systems, point-based representations are temporarily stored in the rendering pipeline to accelerate rendering [21], [22]. We have systematically addressed the problem of representing texture functions on point-sampled objects and avoiding aliasing during rendering in [23]. The surface splatting technique can replace the heuristics used in previous methods and provide superior texture quality.

We develop EWA splatting along similar lines to the seminal work of Heckbert [9], who introduced EWA filtering to avoid aliasing of surface textures. We recently extended his framework to represent and render texture functions on irregularly point-sampled surfaces [23] and to volume splatting [24]. Section 6.4 will show the connection between EWA volume and surface splatting.

3 Ideal Resampling

3.1 Sampling and Aliasing

Aliasing is a fundamental problem in computer graphics. Although, conceptually, computer graphics often deals with continuous representations of graphics models, in practice, computer-generated images are represented by a discrete array of samples. Image synthesis involves the conversion between continuous and discrete representations, which may cause aliasing artifacts such as moiré patterns and jagged edges, illustrated in Fig. 1, or flickering in animations.

To study aliasing, it is useful to interpret images, surface textures, or volume data as multidimensional signals. In the following discussion, we will focus on one-dimensional signals and return to multidimensional signals in Sections 4 and 5. When a continuous signal is converted to a discrete signal it is evaluated, or *sampled*, on a discrete grid. To analyze the effects of sampling and to understand the relation between the continuous and the discrete representation of

a signal, we review some definitions and results from signal processing theory.

A *filter* is a process that takes a signal as an input and generates a modified signal or a *response* as an output. The easiest class of filters to understand are *linear space invariant* filters. A linear space invariant filter $\mathcal{L}$ is uniquely characterized by its *impulse response* $h(x)$, i.e., its output resulting from an impulse input. As a consequence, the response of a linear space invariant filter to any input signal $f(x)$ is given by the *convolution* of $f(x)$ and $h(x)$:

$$\mathcal{L}\{f(x)\} = \int_{-\infty}^{+\infty} f(t)h(x-t)dt = (f \otimes h)(x).$$

A fundamental approach to analyze a filter is to compute its eigenfunctions and eigenvalues. The eigenfunctions of linear time invariant filters are *complex exponentials* and the eigenvalues are given by the Fourier transform of its impulse response, which is called *frequency response*. The Fourier transform of a signal $f(x)$ is called the *spectrum* of the signal, denoted by $F(\omega)$, where ω is the angular frequency. We write $f(x) \leftrightarrow F(\omega)$ to relate the *spatial* and the *frequency domain* representation of the signal. One of the most useful properties of the Fourier transform is that the Fourier transform of the convolution of two signals is the product of their Fourier transforms, i.e., $f \otimes g \leftrightarrow FG$ and vice versa, i.e., $fg \leftrightarrow F \otimes G/2\pi$.

We analyze the sampling of a continuous signal using the Fourier transform and frequency domain representations, shown in Fig. 2. Sampling a continuous signal $a_c(x)$ is performed by multiplying it with an impulse train $i(x)$, which is a sum of unit-spaced impulses, i.e., $i(x) = \sum_n \delta(x-n)$ (Fig. 2b). This yields the discrete signal $a(x) = a_c(x)i(x/T)$, where T is the sample distance. In the frequency domain, this results in the spectrum of the discrete signal $A(\omega)$ given by the convolution $A(\omega) = A_c(\omega) \otimes I(\omega)/2\pi$. Since the Fourier transform of the impulse train $i(x/T)$ is another impulse train $I(\omega) = \omega_s i(\omega/\omega_s)$, $\omega_s = 2\pi/T$, the spectrum of the discrete signal consists of a superposition of replicas of the spectrum of the continuous signal spaced at a distance ω_s (Fig. 2c).

To reconstruct the continuous signal, we have to eliminate all replicas of A_c from A except the central one. If the replicas do not overlap, this is achieved by multiplying $A(\omega)$ with a box function $H^{\omega_s}(\omega) = 1$ for $\omega \leq \omega_s$ and 0 otherwise. H^{ω_s} is called an *ideal low-pass filter* with cutoff frequency ω_s, where $\omega_s/2$ is also called the Nyquist frequency of the sampling grid. In the spatial domain, the impulse response of H^{ω_s} is a sinc function. However, if the maximum frequency ω_a in the spectrum of A_c is higher than ω_s as shown in Fig. 2, the replicas overlap and it is impossible to reconstruct the original spectrum A_c from A (Fig. 2c). High frequencies from the replicas appear as low frequencies in the original spectrum (Fig. 2e), which is called *aliasing*.

3.2 Antialiasing

From the above discussion, we conclude that there are two approaches to reduce aliasing problems: We can either sample the continuous signal at a higher frequency or we eliminate frequencies above the Nyquist limit before sampling, which is called *prefiltering*. Since most signals of interest are not band limited, sampling at a higher frequency will alleviate, but not completely avoid, aliasing. Moreover, increasing the sampling frequency leads to higher memory and computational requirements of most algorithms. On the other hand, prefiltering is performed by applying a low-pass filter to the signal before sampling, hence it is the more theoretically justified antialiasing method. Using an ideal low-pass filter with cutoff frequency $\omega_s/2$, the filtered signal will be band limited to the Nyquist frequency of the sampling grid and, thus, it can be reconstructed exactly. In practice, prefiltering is implemented as a convolution in the spatial domain, hence prefilters with a small support are desirable for efficiency reasons. However, the widths of a filter in the spatial and frequency domains are inversely related; therefore, some aliasing will be inevitable during sampling.

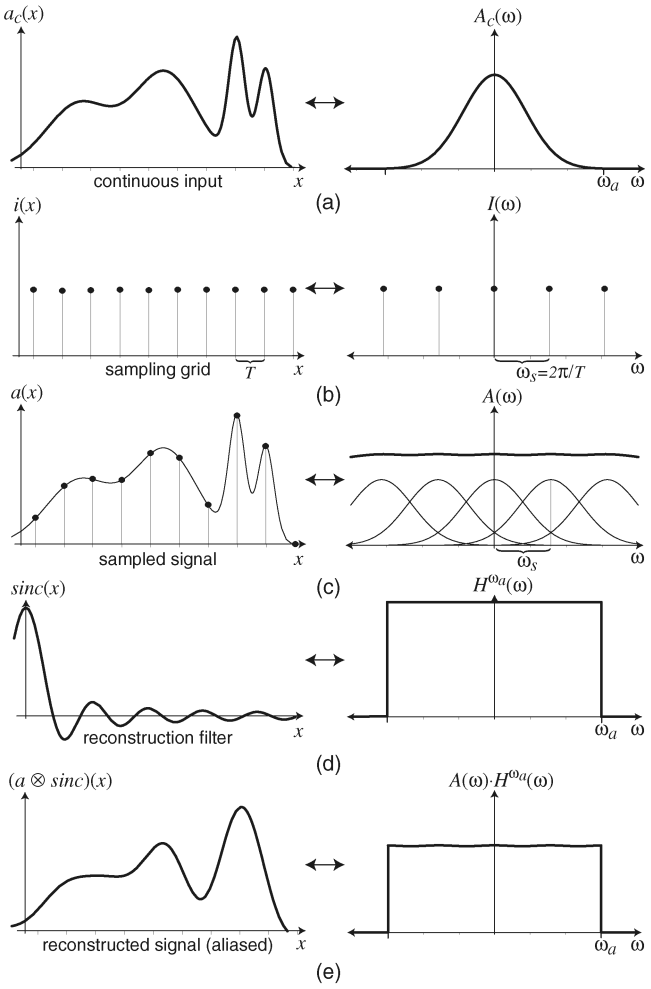


Fig. 2. Frequency analysis of aliasing.

3.3 Rendering and Ideal Resampling Filters

In our framework, graphics models are represented as a set of irregularly spaced samples of multidimensional functions describing object attributes such as volume opacity (Section 4) or surface textures (Section 5). We reconstruct the continuous attribute functions by computing a weighted sum

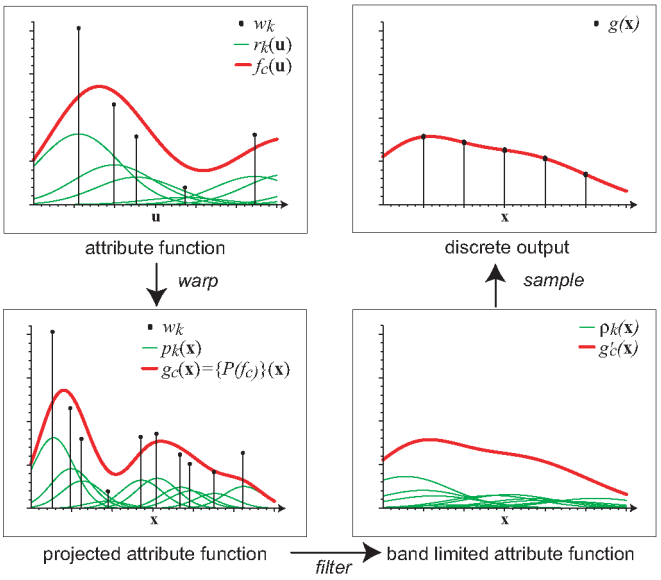


Fig. 3. Projection, filtering, and sampling of a 1D attribute function.

$$f_c(\mathbf{u}) = \sum_{k \in \mathbb{N}} w_k r_k(\mathbf{u}), \tag{1}$$

where r_k is called a reconstruction kernel centered at the sample position $\mathbf{u}_k$ and w_k is a sample value, e.g., the diffuse color at $\mathbf{u}_k$. We use the term *source space* to denote the domain of $f_c(\mathbf{u})$.

We interpret rendering an attribute function (1) as a resampling process, involving the three steps illustrated in Fig. 3:

1. Project $f_c(\mathbf{u})$ from source to screen space, yielding the continuous screen space signal $g_c(\mathbf{x})$:

$$g_c(\mathbf{x}) = \{\mathcal{P}(f_c)\}(\mathbf{x}), \tag{2}$$

where $\mathbf{x}$ are 2D screen space coordinates and projection is denoted by the projection operator $\mathcal{P}$. Note that the operators $\mathcal{P}$ used for rendering (Sections 4 and 5) are linear in their arguments (however, this does *not* imply that the projection performed by $\mathcal{P}$ is a linear mapping). Therefore, we can reformulate (2) by first projecting the reconstruction kernels before computing the sum:

$$g_c(\mathbf{x}) = \left\{ \mathcal{P}\left(\sum_{k \in \mathbb{N}} w_k r_k \right) \right\}(\mathbf{x}) = \sum_{k \in \mathbb{N}} w_k p_k(\mathbf{x}), \tag{3}$$

introducing the abbreviation $p_k = \mathcal{P} r_k$ for the projected reconstruction kernels.

2. Band limit the screen space signal using a prefilter h, resulting in the continuous output function $g'_c(\mathbf{x})$:

$$g'_c(\mathbf{x}) = g_c(\mathbf{x}) \otimes h(\mathbf{x}) = \int_{\mathbb{R}^2} g_c(\boldsymbol{\eta}) h(\mathbf{x} - \boldsymbol{\eta}) d\boldsymbol{\eta}. \tag{4}$$

3. Sample the continuous output function by multiplying it with an impulse train i to produce the discrete output $g(\mathbf{x})$:

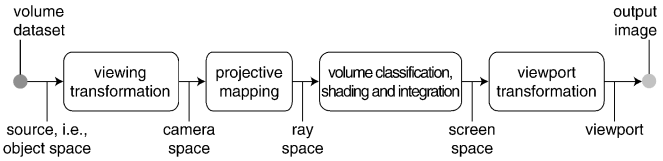


Fig. 4. The forward mapping volume rendering pipeline.

$$g(\mathbf{x}) = g'_c(\mathbf{x}) i(\mathbf{x}).$$

An explicit expression for the projected continuous output function can be derived by expanding the above relations in reverse order:

$$\begin{aligned} g'_c(\mathbf{x}) &= \int_{\mathbb{R}^2} \left\{ \mathcal{P}\left(\sum_{k \in \mathbb{N}} w_k r_k \right) \right\}(\boldsymbol{\eta}) h(\mathbf{x} - \boldsymbol{\eta}) \, d\boldsymbol{\eta} \\ &= \sum_{k \in \mathbb{N}} w_k \int_{\mathbb{R}^2} p_k(\boldsymbol{\eta}) h(\mathbf{x} - \boldsymbol{\eta}) \, d\boldsymbol{\eta} \\ &= \sum_{k \in \mathbb{N}} w_k \rho_k(\mathbf{x}), \end{aligned} \tag{5}$$

$$\text{where} \quad \rho_k(\mathbf{x}) = (p_k \otimes h)(\mathbf{x}). \tag{6}$$

We call a projected and filtered reconstruction kernel $\rho_k(\mathbf{x})$ an *ideal resampling kernel*, which is expressed here as a convolution in screen space. Exploiting the linearity of the projection operator, (5) states that we can first project and filter each reconstruction kernel r_k individually to derive the resampling kernels ρ_k and then sum up the contributions of these kernels in screen space.

In the following Sections 4 and 5, we will model the rendering process for volume data and for point-sampled surfaces, respectively, as a resampling problem by expressing it in the form of (5) and (6). Since this resampling technique is based on the *prefiltering* approach to antialiasing, it leads to high image quality with few aliasing artifacts, irrespective of the spectrum of the unfiltered screen space signal.

4 Volume Resampling

We distinguish two fundamental approaches to volume rendering: backward mapping algorithms that shoot rays through pixels on the image plane into the volume data and forward mapping algorithms that map the data onto the image plane. In the following discussion, we will describe a forward mapping technique. Mapping the data onto the image plane involves a sequence of intermediate steps where the data is transformed to different coordinate systems, as in conventional rendering pipelines. We introduce our terminology in Fig. 4. Note that the terms *space* and *coordinate system* are synonymous. The figure summarizes a *forward mapping volume rendering pipeline*, where the data flows from the left to the right.

As an overview, we briefly describe the coordinate systems and transformations that are relevant for our technique. We will deal in detail with the effect of the transformations in Section 6.2. The volume data is initially given in source space, which is usually called *object space* in

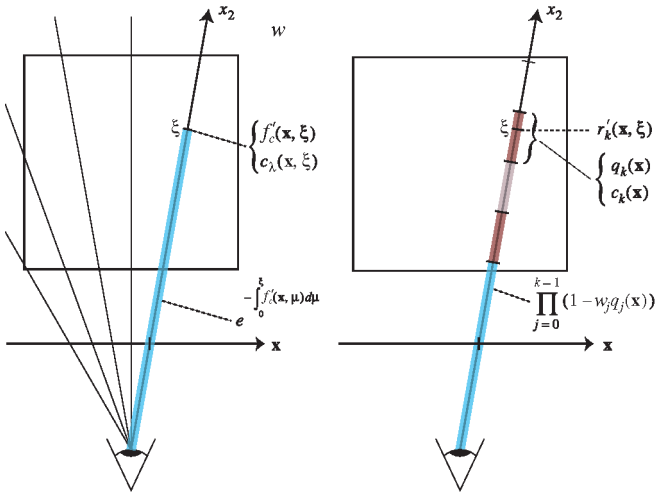


Fig. 5. Volume rendering. Left: Illustrating the volume rendering equation in 2D. Right: Approximation in typical splatting algorithms

this context. To render the data from an arbitrary viewpoint, it is first mapped to *camera space* using the viewing transformation. The camera coordinate system is defined such that its origin is at the center of projection.

We further transform the data to *ray space*, which is introduced in Section 4.1. Ray space is a non-Cartesian coordinate system that enables an easy formulation of the volume rendering equation. In ray space, the viewing rays are parallel to a coordinate axis, facilitating analytical integration of the volume function. We present the transformation from camera to ray space in Section 6.2; it is a key element of our technique. Since its purpose is similar to the projective transform used in rendering pipelines such as OpenGL, it is also called the *projective mapping*.

Evaluating the volume rendering equation results in a 2D image in *screen space*. In a final step, this image is transformed to *viewport coordinates*. Focusing on the essential aspects of our technique, we are not covering the viewport transformation in the following explanations. However, it can be easily incorporated in an implementation. Moreover, we do not discuss volume classification and shading in a forward mapping pipeline, but refer to [25] or [26] for a thorough discussion.

4.1 Splatting Algorithms

We review the low albedo approximation of the volume rendering equation [27], [28] as used for fast, direct volume rendering [2], [29], [25], [30]. The left part of Fig. 5 illustrates the corresponding situation in 2D. Starting from this form of the rendering equation, we discuss several simplifying assumptions leading to the well-known *splatting* formulation. Because of their efficiency, splatting algorithms [2], [25] belong to the most popular forward mapping volume rendering techniques.

We slightly modify the conventional notation, introducing our concept of ray space. We denote a point in ray space by a column vector of three coordinates $\mathbf{x} = (x_0, x_1, x_2)^T$. Given a center of projection and a projection plane, these three coordinates are interpreted geometrically as follows: The coordinates x_0 and x_1 specify a point on the projection plane. The ray intersecting the center of projection and the point (x_0, x_1) on the projection plane is called a viewing ray. Using the abbreviation $\mathbf{x} = (x_0, x_1)^T$, we also refer to the viewing ray passing through (x_0, x_1) as $\mathbf{x}$. The third coordinate x_2 specifies the Euclidean distance from the center of projection to a point on the viewing ray. Note that our notation does not distinguish between a ray $\mathbf{x}$ and a point in ray space $\mathbf{x}$; however, it will be clear from the context which one is meant. Furthermore, to simplify the notation, we will use any of the synonyms $\mathbf{x}$, $(\mathbf{x}, x_2)^T$, or $(x_0, x_1, x_2)^T$ to denote a point in ray space.

The volume rendering equation describes the light intensity $I_\lambda(\mathbf{x})$ at wavelength λ that reaches the center of projection along the ray $\mathbf{x}$ with length L:

$$I_\lambda(\mathbf{x}) = \int_0^L c_\lambda(\mathbf{x}, \xi) f'_c(\mathbf{x}, \xi) e^{-\int_0^\xi f'_c(\mathbf{x}, \mu)\, d\mu}\, d\xi, \tag{7}$$

where $f'_c(\mathbf{x})$ is the *extinction function* that defines the rate of light occlusion and $c_\lambda(\mathbf{x})$ is an emission coefficient. The exponential term can be interpreted as an *attenuation factor*. Finally, the product $c_\lambda(\mathbf{x}) f'_c(\mathbf{x})$ is also called the *source term* [28], describing the light intensity scattered in the direction of the ray $\mathbf{x}$ at the point x_2. In the following equations, we will omit the parameter λ, implying that (7) has to be evaluated separately for different wavelengths.

Now, we assume that the extinction function in *object space* (i.e., source space) $f_c(\mathbf{u})$ is given in the form of (1) as a weighted sum of coefficients w_k and reconstruction kernels $r_k(\mathbf{u})$. This corresponds to a physical model where the volume consists of individual particles that absorb and emit light. The reconstruction kernels r_k reflect the position and shape of individual particles. The particles can be irregularly spaced and may differ in shape; hence, the model is not restricted to regular data sets. Note that the extinction function in ray space $f'_c(\mathbf{x})$ is computed by concatenating a mapping φ from object space to camera space and a mapping ϕ from camera space to ray space (see Fig. 4), yielding:

$$f'_c(\mathbf{x}) = f_c(\varphi^{-1}(\phi^{-1}(\mathbf{x}))) = \sum_k w_k r'_k(\mathbf{x}), \tag{8}$$

where $r'_k(\mathbf{x}) = r_k(\varphi^{-1}(\phi^{-1}(\mathbf{x})))$ is a reconstruction kernel in ray space. The mappings ϕ and φ will be discussed in detail in Section 6.2.

Because of the linearity of integration, substituting (8) into (7) yields

$$I(\mathbf{x}) = \sum_k w_k \Bigg(\int_0^L c(\mathbf{x}, \xi) r'_k(\mathbf{x}, \xi) \prod_j e^{-w_j \int_0^\xi r'_j(\mathbf{x}, \mu)\, d\mu}\, d\xi \Bigg), \tag{9}$$

which can be interpreted as a weighted sum of projected reconstruction kernels. So, in terms of (3), we have the correspondence $I = \sum_k w_k p_k = g_c$ and, for consistency with Section 3, we will use g_c from now on.

To compute g_c numerically, splatting algorithms make several simplifying assumptions, illustrated in the right part of Fig. 5. Usually, the reconstruction kernels $r'_k(\mathbf{x})$ have local support. The splatting approach assumes that these local

support areas do not overlap along a ray $\mathbf{x}$ and the reconstruction kernels are ordered front to back. We also assume that the emission coefficient is constant in the support of each reconstruction kernel along a ray; hence, we use the notation $c_k(x_0, x_1) = c(x_0, x_1, x_2)$, where (x_0, x_1, x_2) is in the support of r'_k. Moreover, we approximate the exponential function with the first two terms of its Taylor expansion, thus $e^{-x} \approx 1 - x$. Finally, we ignore self-occlusion. Exploiting these assumptions, we rewrite (9), yielding:

$$g_c(\mathbf{x}) = \sum_k w_k c_k(\mathbf{x}) q_k(\mathbf{x}) \prod_{j=0}^{k-1} \big(1 - w_j q_j(\mathbf{x})\big), \tag{10}$$

where $q_k(\mathbf{x})$ denotes an integrated reconstruction kernel, hence:

$$q_k(\mathbf{x}) = \int_{\mathbb{R}} r'_k(\mathbf{x}, x_2)\, dx_2. \tag{11}$$

Equation (10) is the basis for all splatting algorithms. Westover [2] introduced the term *footprint function* for the integrated reconstruction kernels q_k. The footprint function is a 2D function that specifies the contribution of a 3D kernel to each point on the image plane. Since integrating a volume along a viewing ray is analogous to projecting a point on a surface onto the image plane, the coordinates $\mathbf{x} = (x_0, x_1)^T$ are also called *screen coordinates* and we say that $g_c(\mathbf{x})$ and $q_k(\mathbf{x})$ are defined in *screen space*.

Splatting is attractive because of its efficiency, which it derives from the use of preintegrated reconstruction kernels. Therefore, during volume integration, each sample point along a viewing ray is computed using a 2D convolution. In contrast, ray-casting methods require a 3D convolution for each sample point. This provides splatting algorithms with an inherent advantage in rendering efficiency. Moreover, splatting facilitates the use of higher quality kernels with a larger extent than the trilinear kernels typically employed by ray-casting. On the other hand, basic splatting methods are plagued by artifacts because of incorrect visibility determination. This problem is unavoidably introduced by the assumption that the reconstruction kernels do not overlap and are ordered back to front. It has been successfully addressed by several authors, as mentioned in Section 2. In contrast, our main contribution is a novel splat primitive that provides high quality antialiasing and efficiently supports elliptical kernels. We believe that our novel primitive is compatible with all state-of-the-art splatting algorithms.

4.2 The Volume Resampling Filter

The splatting equation (10) represents the output image as a *continuous* screen space signal $g_c(\mathbf{x})$. In order to properly sample this function to a *discrete* output image without aliasing artifacts, it has to be band limited to match the Nyquist frequency of the discrete image. According to (4), we achieve this band limitation by convolving $g_c(\mathbf{x})$ with an appropriate low-pass filter $h(\mathbf{x})$, yielding the *antialiased* splatting equation

$$g'_c(\mathbf{x}) = (g_c \otimes h)(\mathbf{x}) = \sum_k w_k \int_{\mathbb{R}^2} c_k(\eta) q_k(\eta) \prod_{j=0}^{k-1} \big(1 - w_j q_j(\eta)\big) h(\mathbf{x} - \eta)\, d\eta. \tag{12}$$

Unfortunately, the convolution integral in (12) cannot be computed explicitly because of the emission and attenuation terms. Hence, we make two simplifying assumptions to rearrange it, leading to an approximation that can be evaluated efficiently.

First, we assume that the emission coefficient is approximately constant in the support of each footprint function q_k, hence $c_k(\mathbf{x}) \approx c_k$ for all $\mathbf{x}$ in the support area. Together with the assumption that the emission coefficient is constant in the support of each reconstruction kernel along a *viewing ray*, this means that the emission coefficient is constant in the complete *3D support* of each reconstruction kernel. In other words, this corresponds to per-voxel evaluation of the shading model or preshading [25], ignoring the effect of shading for antialiasing. Note that prefiltering methods for surface textures usually ignore aliasing due to shading, too.

Additionally, we assume that the attenuation factor has an approximately constant value o_k in the support of each footprint function. Hence:

$$\prod_{j=0}^{k-1} \big(1 - w_j q_j(\mathbf{x})\big) \approx o_k \tag{13}$$

for all $\mathbf{x}$ in the support area. A variation of the attenuation factor indicates that the footprint function is partially covered by a more opaque region in the volume data. Therefore, this variation can be interpreted as a "soft" edge. Ignoring such situations means that we cannot prevent edge aliasing. Again, this is similar to rendering surfaces, where edge and texture aliasing are handled by different algorithms as well.

Exploiting these simplifications, we can rewrite (12) to:

$$\begin{aligned}(g_c \otimes h)(\mathbf{x}) &\approx \sum_k w_k c_k o_k \int_{\mathbb{R}^2} q_k(\eta) h(\mathbf{x} - \eta)\, d\eta \\ &= \sum_k w_k c_k o_k (q_k \otimes h)(\mathbf{x}).\end{aligned}$$

Following the terminology of Section 3.3 (see (6)), we call

$$\rho_k(\mathbf{x}) = c_k o_k (q_k \otimes h)(\mathbf{x}) = (p_k \otimes h)(\mathbf{x}) \tag{14}$$

an *ideal volume resampling filter*, combining a projected reconstruction kernel $p_k = c_k o_k q_k$ and a low-pass kernel h. Hence, we can approximate the antialiased splatting equation (12) by replacing the footprint function q_k in the original splatting equation (10) with the resampling filter ρ_k. This means that, instead of band limiting the output function $g_c(\mathbf{x})$ directly, we band limit each footprint function separately. Under the assumptions described above, we get a splatting algorithm that produces a band limited output function respecting the Nyquist frequency of the raster image, therefore avoiding aliasing artifacts. Remember that the reconstruction kernels are integrated in ray space, resulting in footprint functions that vary

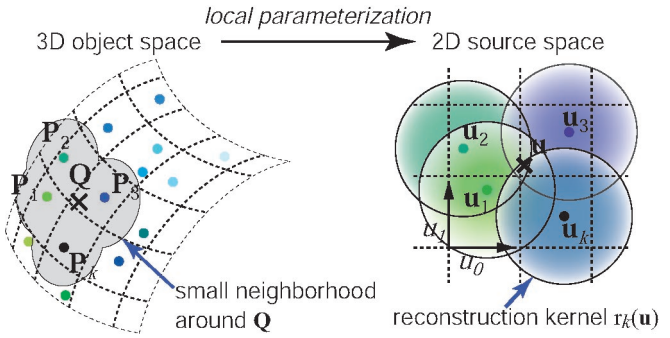


Fig. 6. Defining a texture function on the surface of a point-based object.

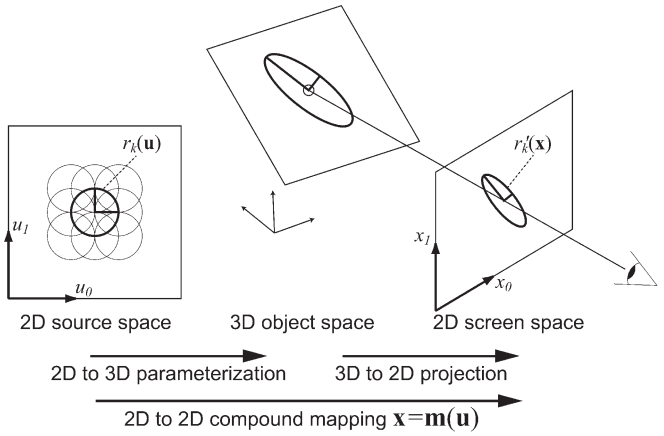


Fig. 7. Mapping a surface function from parameter space to screen space.

significantly in size and shape across the volume. Hence, the resampling filter in (14) is strongly *space variant*.

Swan et al. presented an antialiasing technique for splatting [18] that is based on a uniform scaling of the reconstruction kernels to band limit the extinction function. Their technique produces similar results as our method for radially symmetric kernels. However, for more general kernels, e.g., elliptical kernels, uniform scaling is a poor approximation of ideal low-pass filtering. Aliasing artifacts cannot be avoided without introducing additional blurriness. On the other hand, our method provides nonuniform scaling in these cases, leading to superior image quality, as illustrated in Section 8. Moreover, our analysis above shows that band limiting the extinction function does not guarantee alias free images. Because of shading and edges, frequencies above the Nyquist limit persist. However, these effects are not discussed in [18].

5 Surface Resampling

When rendering point-sampled surfaces, the data flows through a similar pipeline as in forward mapping volume rendering (Fig. 4). In Section 5.1, we first explain how continuous attribute functions are conceptually defined on point-sampled surfaces. Then we introduce an expression similar to (7) for the continuous output function (i.e., the rendered image) in Section 5.2.

5.1 Attribute Functions on Point-Sampled Surfaces

We represent point-sampled surfaces as a set of irregularly spaced points $\{\mathbf{P}_k\}$ in three-dimensional object space without connectivity. A point $\mathbf{P}_k$ has a position and a normal. It is associated with a reconstruction kernel r_k and samples of the attribute functions, e.g., w_k^r, w_k^g, w_k^b that represent continuous functions for red, green, and blue color components. Without loss of generality, we perform all further calculations with scalar coefficients w_k. Note that the basis functions r_k and coefficients w_k can be determined in a preprocessing step as described in [23].

We define a continuous function on the surface represented by the set of points, as illustrated in Fig. 6. Given a point $\mathbf{Q}$ anywhere on the surface in object space, shown left, we construct a local parameterization of the surface in a small neighborhood of $\mathbf{Q}$, illustrated on the right. The points $\mathbf{Q}$ and $\mathbf{P}_k$ have *local source space coordinates* $\mathbf{u}$ and $\mathbf{u}_k$, respectively. Using the parameterization, we can define the continuous attribute function $f_c(\mathbf{u})$ on the surface as in (1):

$$f_c(\mathbf{u}) = \sum_{k \in \mathbb{N}} w_k r_k(\mathbf{u}). \tag{15}$$

We will choose basis functions r_k that have *local* support or that are appropriately truncated. Then, $\mathbf{u}$ lies in the support of a small number of basis functions. Note that, in order to evaluate (15), the local parameterization has to be established in the union of these support areas only, which is very small. Furthermore, we will compute these local parameterizations on the fly during rendering, as described in Section 7.2.

5.2 The Surface Resampling Filter

Rendering a parameterized surface involves mapping the attribute function $f_c(\mathbf{u})$ from parameter, i.e., source space, to screen space. As illustrated in Fig. 7, we denote this 2D to 2D mapping by $\mathbf{x} = \mathbf{m}(\mathbf{u})$. It is composed of a 2D to 3D parameterization from source to object space and a 3D to 2D projection from object to screen space, which are described in more detail in Section 6.3. Using $\mathbf{m}$, we can write the continuous output function as

$$g_c(\mathbf{x}) = \sum_k w_k c(\mathbf{x}) r'_k(\mathbf{x}), \tag{16}$$

where $r'_k(\mathbf{x}) = r_k(\mathbf{m}^{-1}(\mathbf{x}))$ is a reconstruction kernel mapped to screen space. As in Section 4, c is the emission term arising from shading the surface.

Again we assume that emission is constant in the support of each r'_k in screen space, which is equivalent to per-point shading, therefore ignoring aliasing due to shading. So, the band limited output function according to (4) is

$$(g_c \otimes h)(\mathbf{x}) = \sum_k w_k c_k (r'_k \otimes h)(\mathbf{x}). \tag{17}$$

Similarly as in Section 4.2, we call

$$\rho_k(\mathbf{x}) = c_k(r'_k \otimes h)(\mathbf{x}) = (p_k \otimes h)(\mathbf{x}) \tag{18}$$

an *ideal surface resampling filter*, combining a projected surface reconstruction kernel $p_k(\mathbf{x}) = c_k r_k(\mathbf{m}^{-1}(\mathbf{x}))$ and a low-pass kernel $h(\mathbf{x})$.

6 EWA Resampling Filters

For both volume and surface rendering, we choose elliptical Gaussians as reconstruction kernels and low-pass filters since they provide certain features that are crucial for our technique: Gaussians are closed under affine mappings and convolution and integrating a 3D Gaussian along one coordinate axis results in a 2D Gaussian. These properties enable us to analytically compute the volume and surface resampling filters ((14) and (18), respectively) as a single 2D Gaussian, as will be shown in Sections 6.2 and 6.3. In Section 6.1, we summarize the mathematical features of the Gaussians that are exploited in our derivation in the following sections. More details on Gaussians can be found in Heckbert's master's thesis [9].

6.1 Elliptical Gaussian Filters

We define a 3D elliptical Gaussian $\mathcal{G}^3_{\mathbf{V}}(\mathbf{x}-\mathbf{p})$ centered at a point $\mathbf{p}$ with a variance matrix $\mathbf{V}$ as:

$$\mathcal{G}^3_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{(2\pi)^{3/2}|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}, \tag{19}$$

where $|\mathbf{V}|$ is the determinant of $\mathbf{V}$. In this form, the Gaussian is normalized to a unit integral. In the case of a 3D Gaussian, $\mathbf{V}$ is a symmetric 3×3 matrix and $\mathbf{x}$ and $\mathbf{p}$ are column vectors $(x_0, x_1, x_2)^T$ and $(p_0, p_1, p_2)^T$, respectively. Similarly to (19), an elliptical 2D Gaussian $\mathcal{G}^2_{\mathbf{V}}(\mathbf{x}-\mathbf{p})$ is defined as:

$$\mathcal{G}^2_{\mathbf{V}}(\mathbf{x}-\mathbf{p}) = \frac{1}{2\pi|\mathbf{V}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p})^T\mathbf{V}^{-1}(\mathbf{x}-\mathbf{p})}, \tag{20}$$

where $\mathbf{V}$ is a 2×2 variance matrix and $\mathbf{x}$ and $\mathbf{p}$ are 2D vectors. Note that the normalization factor is different for 2D and 3D Gaussians.

We can easily apply an arbitrary affine mapping $\mathbf{u} = \Phi(\mathbf{x})$ to a Gaussian of any dimension n, denoted by $\mathcal{G}^n_{\mathbf{V}}$. Let us define the affine mapping as $\Phi(\mathbf{x}) = \mathbf{M}\mathbf{x} + \mathbf{c}$, where $\mathbf{M}$ is an $n\times n$ matrix and $\mathbf{c}$ is a vector $(c_1, \ldots, c_n)^T$. We substitute $\mathbf{x} = \Phi^{-1}(\mathbf{u})$, yielding:

$$\mathcal{G}^n_{\mathbf{V}}(\Phi^{-1}(\mathbf{u}) - \mathbf{p}) = \frac{1}{|\mathbf{M}^{-1}|}\mathcal{G}^n_{\mathbf{M}\mathbf{V}\mathbf{M}^T}(\mathbf{u} - \Phi(\mathbf{p})). \tag{21}$$

Moreover, convolving two Gaussians with variance matrices $\mathbf{V}$ and $\mathbf{Y}$ results in another Gaussian with variance matrix $\mathbf{V}+\mathbf{Y}$:

$$(\mathcal{G}^n_{\mathbf{V}} \otimes \mathcal{G}^n_{\mathbf{Y}})(\mathbf{x}-\mathbf{p}) = \mathcal{G}^n_{\mathbf{V}+\mathbf{Y}}(\mathbf{x}-\mathbf{p}). \tag{22}$$

Finally, integrating a normalized 3D Gaussian $\mathcal{G}^3_{\mathbf{V}}$ along one coordinate axis yields a normalized 2D Gaussian $\mathcal{G}^2_{\hat{\mathbf{V}}}$, hence:

$$\int_{\mathbb{R}} \mathcal{G}^3_{\mathbf{V}}(\mathbf{x}-\mathbf{p})\, dx_2 = \mathcal{G}^2_{\hat{\mathbf{V}}}(\hat{\mathbf{x}}-\hat{\mathbf{p}}), \tag{23}$$

where $\hat{\mathbf{x}} = (x_0, x_1)^T$ and $\hat{\mathbf{p}} = (p_0, p_1)^T$. The 2×2 variance matrix $\hat{\mathbf{V}}$ is easily obtained from the 3×3 matrix $\mathbf{V}$ by skipping the third row and column:

$$\mathbf{V} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \Leftrightarrow \begin{pmatrix} a & b \\ b & d \end{pmatrix} = \hat{\mathbf{V}}. \tag{24}$$

In the following, we will use $\mathcal{G}_{\mathbf{V}}$ to denote both 2D and 3D Gaussians, with the context clarifying which one is meant.

6.2 The EWA Volume Resampling Filter

In this section, we first describe how to map arbitrary elliptical Gaussian volume reconstruction kernels from object to ray space. Our derivation results in an analytic expression for the kernels in ray space $r'_k(\mathbf{x})$ as in (8). We will then be able to analytically integrate the kernels according to (11) and to convolve the footprint function q_k with a Gaussian low-pass filter h as in (22), yielding an elliptical Gaussian resampling filter ρ_k.

6.2.1 The Viewing Transformation

The reconstruction kernels are initially given in source space, or *object space*, which has coordinates $\mathbf{u} = (u_0, u_1, u_2)^T$. As in Section 4.1, we denote the Gaussian reconstruction kernels in object space by:

$$r_k(\mathbf{u}) = \mathcal{G}_{\mathbf{V}_k}(\mathbf{u} - \mathbf{u}_k), \tag{25}$$

where $\mathbf{u}_k$ are the voxel positions in object space. For regular volume datasets, the variance matrices $\mathbf{V}_k$ are usually identity matrices. For rectilinear datasets, they are diagonal matrices, where the matrix elements contain the squared distances between voxels along each coordinate axis. Curvilinear and irregular grids have to be resampled to a more regular structure in general. For example, Mao et al. [31] describe a stochastic sampling approach with a method to compute the variance matrices for curvilinear volumes.

We denote camera coordinates by a vector $\mathbf{t} = (t_0, t_1, t_2)^T$. Object coordinates are transformed to camera coordinates using a mapping $\mathbf{t} = \varphi(\mathbf{u})$, called *viewing transformation*. The viewing transformation is usually an affine mapping defined by a matrix $\mathbf{W}$ and a translation vector $\mathbf{d}$ as $\varphi(\mathbf{u}) = \mathbf{W}\mathbf{u} + \mathbf{d}$.

6.2.2 The Projective Transformation

We will concatenate the viewing transformation with a projective transformation that converts camera coordinates to ray coordinates, as illustrated in Fig. 8. Camera space is defined such that the origin of the camera coordinate system is at the center of projection and the projection plane is the plane $t_2 = 1$. Camera space and ray space are related by the mapping $\mathbf{x} = \phi(\mathbf{t})$. Using the definition of ray space from Section 4.1, $\phi(\mathbf{t})$ and its inverse $\phi^{-1}(\mathbf{t})$ are therefore given by:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \phi(\mathbf{t}) = \begin{pmatrix} t_0/t_2 \\ t_1/t_2 \\ \|(t_0, t_1, t_2)^T\| \end{pmatrix} \tag{26}$$

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \end{pmatrix} = \phi^{-1}(\mathbf{x}) = \begin{pmatrix} x_0/l \cdot x_2 \\ x_1/l \cdot x_2 \\ 1/l \cdot x_2 \end{pmatrix}, \tag{27}$$

where $l = \|(x_0, x_1, 1)^T\|$.

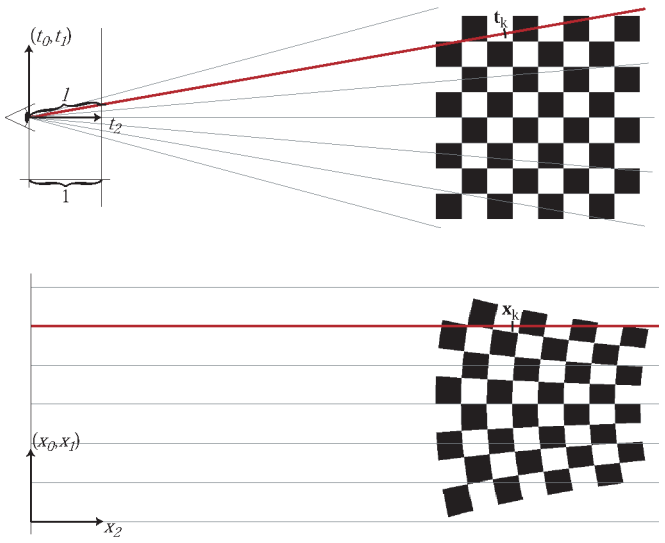


Fig. 8. Transforming the volume from camera to ray space. Top: camera space. Bottom: ray space.

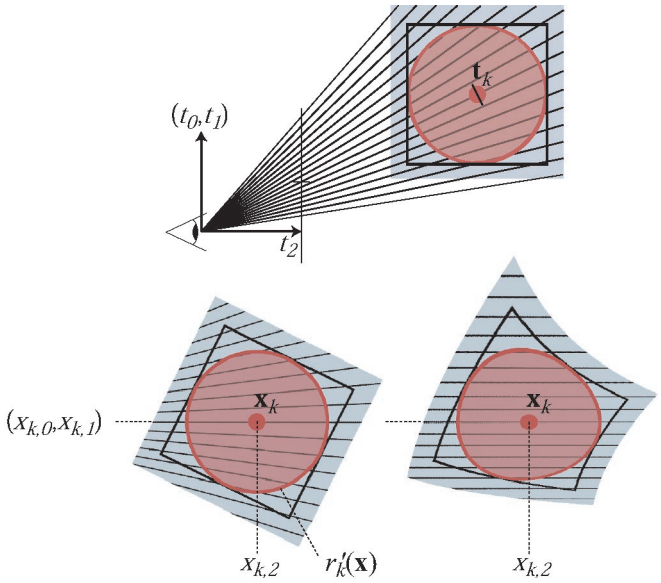


Fig. 9. Mapping a reconstruction kernel from camera to ray space. Top: camera space. Bottom: ray space. Left: local affine mapping. Right: exact mapping.

Unfortunately, these mappings are not affine, so we cannot apply (21) directly to transform the reconstruction kernels from camera to ray space. To solve this problem, we introduce the *local affine approximation* ϕ_k of the projective transformation. It is defined by the first two terms of the Taylor expansion of ϕ at the point $\mathbf{t}_k$:

$$\phi_k(\mathbf{t}) = \mathbf{x}_k + \mathbf{J}_k \cdot (\mathbf{t} - \mathbf{t}_k), \tag{28}$$

where $\mathbf{t}_k$ is the center of a Gaussian in camera space and $\mathbf{x}_k = \phi(\mathbf{t}_k)$ is the corresponding position in ray space. The Jacobian $\mathbf{J}_k$ is given by the partial derivatives of ϕ at the point $\mathbf{t}_k$:

$$\mathbf{J}_k = \frac{\partial \phi}{\partial \mathbf{t}}(\mathbf{t}_k) = \begin{pmatrix} 1/t_{k,2} & 0 & -t_{k,0}/t_{k,2}^2 \\ 0 & 1/t_{k,2} & -t_{k,1}/t_{k,2}^2 \\ t_{k,0}/l' & t_{k,1}/l' & t_{k,2}/l' \end{pmatrix}, \tag{29}$$

where $l' = \|(t_{k,0}, t_{k,1}, t_{k,2})^T\|$.

The local affine approximation of the compound mapping from source to ray space $\mathbf{x} = \mathbf{m}_k(\mathbf{u})$ is given by the concatenation of $\mathbf{t} = \varphi(\mathbf{u})$ and $\mathbf{x} = \phi_k(\mathbf{t})$:

$$\begin{aligned} \mathbf{x} = \mathbf{m}_k(\mathbf{u}) &= \phi_k(\varphi(\mathbf{u})) \\ &= \mathbf{J}_k\mathbf{W}\mathbf{u} + \mathbf{x}_k + \mathbf{J}_k(\mathbf{d} - \mathbf{t}_k). \end{aligned}$$

We substitute $\mathbf{u} = \mathbf{m}_k^{-1}(\mathbf{x})$ in (25) and apply (21) to map the reconstruction kernels to ray space, yielding the desired expression for $r'_k(\mathbf{x})$:

$$\begin{aligned} r'_k(\mathbf{x}) &= \mathcal{G}_{\mathbf{V}_k}(\mathbf{m}^{-1}(\mathbf{x}) - \mathbf{u}_k) \\ &= \frac{1}{|\mathbf{W}^{-1}\mathbf{J}_k^{-1}|}\mathcal{G}_{\mathbf{V}'_k}(\mathbf{x} - \mathbf{m}(\mathbf{u}_k)), \end{aligned} \tag{30}$$

where $\mathbf{V}'_k$ is the variance matrix in ray coordinates. According to (21), $\mathbf{V}'_k$ is given by:

$$\mathbf{V}'_k = \mathbf{J}_k\mathbf{W}\mathbf{V}_k\mathbf{W}^T\mathbf{J}_k^T. \tag{31}$$

Note that, for uniform or rectilinear datasets, the product $\mathbf{W}\mathbf{V}_k\mathbf{W}^T$ has to be computed only once per frame since $\mathbf{V}_k$ is the same for all voxels and $\mathbf{W}$ only changes from frame to frame. Since the Jacobian is different for each voxel position, $\mathbf{V}'_k$ has to be recalculated for each voxel, requiring two 3×3 matrix multiplications $\mathbf{V}'_k = \mathbf{J}_k(\mathbf{W}\mathbf{V}_k\mathbf{W}^T)\mathbf{J}_k^T$. In the case of curvilinear or irregular volumes, each reconstruction kernel has an individual variance matrix $\mathbf{V}_k$. Our method efficiently handles this situation, requiring only one additional 3×3 matrix multiplication, i.e., $\mathbf{V}'_k = (\mathbf{J}_k\mathbf{W})\mathbf{V}_k(\mathbf{J}_k\mathbf{W})^T$. In contrast, previous techniques [2], [31] cope with elliptical kernels by computing their projected extents in screen space and then establishing a mapping to a circular footprint table. However, this procedure is computationally expensive. It leads to a bad approximation of the integral of the reconstruction kernel, as pointed out in [14], [18].

As illustrated in Fig. 9, the local affine mapping is exact only for the ray passing through $\mathbf{t}_k$ or $\mathbf{x}_k$, respectively. The figure is exaggerated to show the nonlinear effects in the exact mapping. The affine mapping essentially approximates the perspective projection with an oblique orthographic projection. Therefore, parallel lines are preserved and approximation errors grow with increasing ray divergence. However, the errors do not lead to visual artifacts in general [14] since the fan of rays intersecting a reconstruction kernel has a small opening angle due to the local support of the reconstruction kernels.

A common approach of performing splatting with perspective projection is to map the footprint function onto a *footprint polygon* in camera space in a first step. In the next step, the footprint polygon is projected to screen space and rasterized, resulting in the so-called *footprint image*. As mentioned in [14], however, this requires significant computational effort. In contrast, our framework efficiently performs perspective projection by mapping the volume to ray space, which requires only the computation of the Jacobian and two 3×3 matrix multiplications. For spherical reconstruction kernels, these matrix operations can be further optimized, as shown in Section 7.1.

6.2.3 Integration and Band Limiting

We integrate the Gaussian reconstruction kernel of (30) according to (11), resulting in a Gaussian footprint function q_k:

$$\begin{aligned} q_k(\mathbf{x}) &= \int_{\mathbb{R}} \frac{1}{|\mathbf{W}^{-1}\mathbf{J}_k^{-1}|} \mathcal{G}_{\mathbf{V}'_k}(\mathbf{x}-\mathbf{x}_k, x_2 - x_{k2})\, dx_2 \\ &= \frac{1}{|\mathbf{W}^{-1}\mathbf{J}_k^{-1}|} \mathcal{G}_{\hat{\mathbf{V}}'_k}(\mathbf{x}-\mathbf{x}_k), \end{aligned} \tag{32}$$

where the 2×2 variance matrix $\hat{\mathbf{V}}'_k$ of the footprint function is obtained from $\mathbf{V}'_k$ by skipping the last row and column, as shown in (24).

Finally, we choose a Gaussian low-pass filter $h(\mathbf{x}) = \mathcal{G}_{\mathbf{V}^h}(\mathbf{x})$, where the variance matrix $\mathbf{V}^h \in \mathbb{R}^{2\times 2}$ is typically the identity matrix. With (22), we compute the convolution in (14), yielding the *EWA volume resampling filter*, or *EWA volume splat*:

$$\begin{aligned} \rho_k(\mathbf{x}) &= (p_k \otimes h)(\mathbf{x}) \\ &= c_k o_k \frac{1}{|\mathbf{W}^{-1}\mathbf{J}_k^{-1}|} (\mathcal{G}_{\hat{\mathbf{V}}'_k} \otimes \mathcal{G}_{\mathbf{V}^h})(\mathbf{x}-\mathbf{x}_k) \\ &= c_k o_k \frac{1}{|\mathbf{W}^{-1}\mathbf{J}_k^{-1}|} \mathcal{G}_{\hat{\mathbf{V}}'_k + \mathbf{V}^h}(\mathbf{x}-\mathbf{x}_k). \end{aligned} \tag{33}$$

6.3 The EWA Surface Resampling Filter

In this section, we first describe how to construct the local parameterizations that are needed to define surface attribute functions $f_c(\mathbf{u})$ (Section 5.1). Then, we derive a mapping $\mathbf{x} = \mathbf{m}(\mathbf{u})$ involving parameterization, viewing transformation, and perspective projection that transforms the attribute function from source to screen space, similarly to Section 6.2.

6.3.1 Local Surface Parameterizations

At each point $\mathbf{P}_k$, a local surface parameterization ψ_k is defined by two orthogonal unit vectors $\mathbf{e}_k^0$ and $\mathbf{e}_k^1$ in the tangent plane of the surface. The tangent plane is given by the surface normal $\mathbf{n}_k$ stored with each point $\mathbf{P}_k$. Hence, each point $\mathbf{u} = (u_0, u_1)^T$ in the parameter domain corresponds to a point $\hat{\mathbf{u}} = (\hat{u}_0, \hat{u}_1, \hat{u}_2)^T$ on the surface in object space:

$$\hat{\mathbf{u}} = \psi_k(\mathbf{u}) = \mathbf{P}_k + \mathbf{S}_k\mathbf{u},$$

where $\mathbf{S}_k$ is a 3×2 matrix consisting of the column vectors $\mathbf{e}_k^0$ and $\mathbf{e}_k^1$.

We denote the Gaussian surface reconstruction kernels in the parameter domain by $r_k(\mathbf{u}) = \mathcal{G}_{\mathbf{V}_k}(\mathbf{u})$. The variance matrix $\mathbf{V}_k$ has to be chosen appropriately to match the local density of points around $\mathbf{P}_k$. Restricting ourselves to radially symmetric kernels, $\mathbf{V}_k$ is a 2×2 identity matrix $\mathbf{I}$ scaled by a factor σ^2, i.e., $\mathbf{V}_k = \sigma^2\mathbf{I}$. The scaling σ depends on the distance between $\mathbf{P}_k$ and its nearest neighbors, e.g., we choose σ as the average distance to the six nearest neighbors. A more sophisticated analysis of the point distribution around $\mathbf{P}_k$ could be used to find suitable variance matrices of general elliptical kernels.

6.3.2 The Viewing Transformation

The viewing transformation that maps object coordinates $\hat{\mathbf{u}}$ to camera coordinates $\mathbf{t}$ is defined as in Section 6.2, i.e., $\mathbf{t} = \varphi(\hat{\mathbf{u}}) = \mathbf{W}\hat{\mathbf{u}} + \mathbf{d}$.

6.3.3 Perspective Projection

Surface points $\mathbf{t}$ in 3D camera space are projected to 2D screen space $\mathbf{x}$ by dividing by the depth coordinate t_2. Hence, we use the same mapping ϕ (see (26)) as for volumes, except that we do not need the third coordinate x_2:

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \phi(\mathbf{t}) = \begin{pmatrix} t_0/t_2 \\ t_1/t_2 \end{pmatrix}.$$

We use the same local affine approximation ϕ_k as in (28). Note that here the Jacobian $\mathbf{J}_k$ is a 2×3 matrix:

$$\mathbf{J}_k = \frac{\partial \phi}{\partial \mathbf{t}}(\mathbf{t}_k) = \begin{pmatrix} 1/t_{k,2} & 0 & -t_{k,0}/t_{k,2}^2 \\ 0 & 1/t_{k,2} & -t_{k,1}/t_{k,2}^2 \end{pmatrix}. \tag{34}$$

Concatenating ψ_k, φ, and ϕ_k, we get the local affine approximation $\mathbf{m}_k$ of the mapping from source space to screen space:

$$\begin{aligned} \mathbf{x} = \mathbf{m}_k(\mathbf{u}) &= \phi_k(\varphi(\psi_k(\mathbf{u}))) \\ &= \mathbf{J}_k\mathbf{W}\mathbf{S}_k\mathbf{u} + \mathbf{x}_k. \end{aligned}$$

Substituting $\mathbf{u} = \mathbf{m}_k^{-1}(\mathbf{x})$ and applying (21) we get the Gaussian surface reconstruction kernel in screen space:

$$\begin{aligned} r'_k(\mathbf{x}) &= \mathcal{G}_{\mathbf{V}_k}(\mathbf{m}_k^{-1}(\mathbf{x}) - \mathbf{u}_k) \\ &= \frac{1}{|\mathbf{M}_k^{-1}|} \mathcal{G}_{\mathbf{V}'_k}(\mathbf{x} - \mathbf{m}_k(\mathbf{u}_k)), \end{aligned}$$

with the variance matrix $\mathbf{V}'_k = \mathbf{M}_k\mathbf{V}_k\mathbf{M}_k^T$ and $\mathbf{M}_k = \mathbf{J}_k\mathbf{W}\mathbf{S}_k \in \mathbb{R}^2$.

6.3.4 Band Limiting

With a Gaussian low-pass filter $h = \mathcal{G}_{\mathbf{V}^h}$ and using (18), the *EWA surface resampling filter*, or *EWA surface splat*, is:

$$\begin{aligned} \rho_k(\mathbf{x}) &= c_k(r'_k \otimes h)(\mathbf{x}) \\ &= c_k \frac{1}{|\mathbf{M}_k^{-1}|} \mathcal{G}_{\mathbf{V}'_k + \mathbf{V}^h}(\mathbf{x} - \mathbf{m}_k(\mathbf{u}_k)). \end{aligned} \tag{35}$$

In (35), the resampling filter is a function in screen space. Since the mapping $\mathbf{m}_k$ is affine and invertible, we can alternatively express it as a function in source space, too. We use

$$\mathbf{x} - \mathbf{m}_k(\mathbf{u}_k) = \mathbf{J}_k\mathbf{J}_k^{-1}(\mathbf{x} - \mathbf{m}_k(\mathbf{u}_k)) = \mathbf{J}_k(\mathbf{m}_k^{-1}(\mathbf{x}) - \mathbf{u}_k),$$

and substitute this into (25), yielding:

$$\rho_k(\mathbf{x}) = c_k\mathcal{G}_{\mathbf{V}_k + \mathbf{M}_k^{-1}\mathbf{V}^h\mathbf{M}_k^{-1^T}}(\mathbf{u} - \mathbf{u}_k). \tag{36}$$

This is the well-known *source space* EWA method [9] extended for irregular sample positions, which is mathematically equivalent to our screen space formulation. However, (36) involves backward mapping a point $\mathbf{x}$ from screen to the object surface, which is impractical for interactive rendering. It amounts to ray tracing the point cloud to find surface intersections. Additionally, the locations $\mathbf{u}_k$ are irregularly positioned such that the evaluation of the resampling kernel

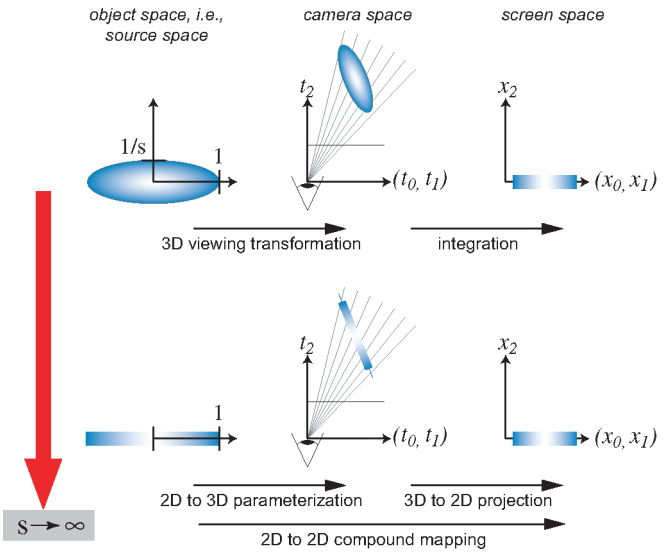


Fig. 10. Reducing a volume reconstruction kernel to a surface reconstruction kernel by flattening the kernel in one dimension. Top: rendering a volume kernel. Bottom: rendering a surface kernel.

in object space is laborious. On the other hand, (35) can be implemented efficiently for point-based objects, as described in Section 7.2.

6.4 Reduction from Volume to Surface Reconstruction Kernels

Since our EWA volume resampling filter can handle arbitrary Gaussian reconstruction kernels, we can represent the structure of a volume data set more accurately by choosing the shape of the reconstruction kernels appropriately. For example, we can improve the precision of isosurface rendering by flattening the reconstruction kernels in the direction of the surface normal. We will show below that an infinitesimally flat Gaussian volume kernel is equivalent to a Gaussian surface texture reconstruction kernel. In other words, we can extract and render a surface representation from a volume data set directly by flattening volume reconstruction kernels into surface reconstruction kernels. Our derivation is illustrated in Fig. 10.

We construct a flattened Gaussian reconstruction kernel in object space by scaling an arbitrary Gaussian with variance matrix $\mathbf{V}$ along the third coordinate axis, i.e., using a scaling matrix $diag(1, 1, 1/s)$. Applying (21), we find that the variance matrix $\mathbf{V}^s$ of the scaled Gaussian is:

$$\mathbf{V}^s = \begin{pmatrix} v_{0,0} & v_{0,1} & v_{0,2}/s^2 \\ v_{1,0} & v_{1,1} & v_{1,2}/s^2 \\ v_{2,0}/s^2 & v_{2,1}/s^2 & v_{2,2}/s^2 \end{pmatrix}.$$

In the limit, if $s = \infty$, $\mathbf{V}^s$ is equivalent to a 2D Gaussian with variance matrix $\hat{\mathbf{V}}$ that is parameterized onto the plane perpendicular to the third coordinate axis, where

$$\hat{\mathbf{V}} = \begin{pmatrix} v_{0,0} & v_{0,1} \\ v_{1,0} & v_{1,1} \end{pmatrix}$$

is the upper left 2×2 matrix in $\mathbf{V}$. Since the plane is defined by basis vectors $(1, 0, 0)^T$ and $(0, 1, 0)^T$, parameterization yields:

```
1: for each voxel k {
2:   compute camera coords. t[k];
3:   compute the Jacobian J[k];
4:   compute the variance matrix V[k];
5:   project t[k] to screen coords. x[k];
6:   setup the resampling filter rho[k];
7:   rasterize and accumulate rho[k];
8: }
```

Fig. 11. The EWA volume splatting algorithm.

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \hat{\mathbf{V}} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \mathbf{V}^s \quad \text{for} \quad s \to \infty. \tag{37}$$

In the limit, the third row and column of $\mathbf{V}^s$ contain only zeros. Therefore, projecting a Gaussian $\mathcal{G}_{\mathbf{V}^s}$ to screen space via mapping to ray space (30) and integration (32), or using perspective projection as in (35), results in the same 2D variance matrix of the reconstruction kernel in screen space. In other words, it is equivalent to rendering the flattened Gaussian as a volume or as a surface reconstruction kernel.

7 Implementation

7.1 EWA Volume Splatting

We implemented a volume rendering algorithm based on the EWA splatting equation. Our implementation is embedded in the VTK (visualization toolkit) framework [32]. We did not optimize our code for rendering speed. We use a sheet buffer to first accumulate splats from planes in the volume that are most parallel to the projection plane [2]. In a second step, the final image is computed by compositing the sheets back to front. Shading is performed using the gradient estimation functionality provided by VTK and the Phong illumination model.

7.1.1 Algorithm

We summarize the main steps that are required to compute the EWA splat for each voxel in Fig. 11.

First, we compute the camera coordinates $\mathbf{t}_k$ of the current voxel k by applying the viewing transformation to the voxel center. Using $\mathbf{t}_k$, we then evaluate the Jacobian $\mathbf{J}_k$ as given in (29). In line 4, we transform the Gaussian reconstruction kernel from object to ray space. This transformation is implemented by (31) and it results in the 3×3 variance matrix $\mathbf{V}'_k$ of the Gaussian in ray space. Remember that $\mathbf{W}$ is the rotational part of the viewing transformation, hence it is typically orthonormal. Additionally, for spherical kernels, $\mathbf{V}_k$ is the identity matrix. In this case, evaluation of (31) can be simplified significantly. Next, we project the voxel center from camera space to the screen by performing a perspective division on $\mathbf{t}_k$. This yields the 2D screen coordinates $\mathbf{x}_k$. Now, we are ready to set up the resampling filter ρ_k according to (33). Its variance matrix is derived from $\mathbf{V}'_k$ by omitting the third row and column and adding a 2×2 identity matrix for the low-pass filter. We compute the determinants $1/|\mathbf{J}_k^{-1}|$ and $1/|\mathbf{W}^{-1}|$ that are used as normalization factors and we evaluate the shading model yielding the emission coefficient c_k.

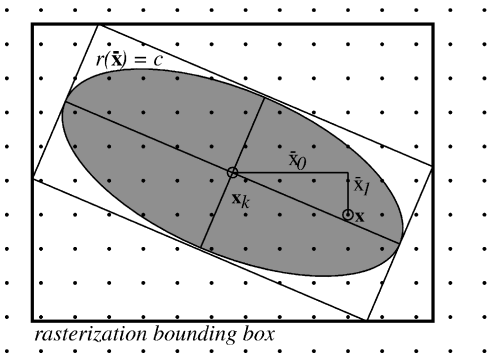


Fig. 12. Rasterizing the resampling filter.

7.1.2 Rasterization

Finally, we rasterize the resampling filter in line 7. As can be seen from the definition of the elliptical Gaussian (19), we also need the inverse of the variance matrix, which is called the *conic matrix*. Let us denote the 2×2 conic matrix of the resampling filter by $\mathbf{Q}$. Furthermore, we define the radial index function

$$r(\bar{\mathbf{x}}) = \bar{\mathbf{x}}^T \mathbf{Q} \bar{\mathbf{x}} \quad \text{where} \quad \bar{\mathbf{x}} = (\bar{x}_0, \bar{x}_1)^T = \mathbf{x} - \mathbf{x}_k.$$

Note that the contours of the radial index, i.e., $r = const.$ are concentric ellipses. For circular kernels, r is the squared distance to the circle center. The exponential function in (19) can now be written as $e^{-\frac{1}{2}r}$. We store this function in a 1D lookup table. To evaluate the radial index efficiently, we use finite differencing. Since r is biquadratic in $\bar{\mathbf{x}}$, we need only two additions to update r for each pixel. We rasterize r in a rectangular, axis-aligned bounding box centered around $\mathbf{x}_k$, as illustrated in Fig. 12. Typically, we use a threshold $c = 4$ and evaluate the Gaussian only if $r(\bar{\mathbf{x}}) < c$. Heckbert provides pseudocode of the rasterization algorithm in [9].

7.2 EWA Surface Splatting

We can perform EWA surface splatting in our volume renderer using flattened volume reconstruction kernels, as described in Section 6.4. We have also implemented a dedicated surface splatting renderer [23]. The EWA surface splatting algorithm essentially proceeds as described in Section 7.1.

However, the depth complexity of a scene is greater than one in general, but only those splats that belong to the visible surface must be accumulated in a pixel. Since back-to-front ordering of unstructured point clouds during rendering is prohibitive, an alternative mechanism is required that separates the contributions of different surfaces. We use a z-buffer approach, computing the z value of the tangent plane at $\mathbf{P}_k$ at each pixel that is covered by the splat. This can be performed efficiently by forward differencing, similar to the visibility splatting approach of [8]. To determine whether a new contribution belongs to the same surface as is already stored in a pixel, the difference between the new z value and the z value stored in the frame buffer is compared to a threshold. If the difference is smaller than the threshold, the contribution is added to the pixel. Otherwise, given that it is closer to the eye-point, the data of the frame buffer is replaced by the new contribution. It is straightforward to extend this approach to a multilayered z-buffer [33] (similar to an A-buffer [34]) that allows the display of semitransparent surfaces and edge antialiasing [23].

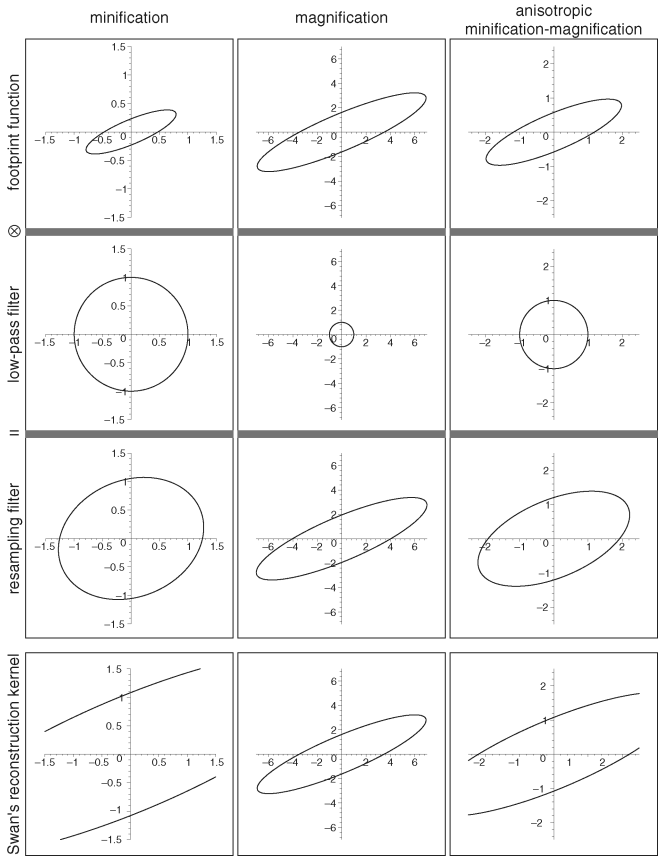


Fig. 13. Properties of the EWA resampling filter.

8 Results

The EWA resampling filter has a number of useful properties, as illustrated in Fig. 13. When the projection to screen space minifies the attribute function (i.e., the volume or point-sampled surface), size and shape of the resampling filter are dominated by the low-pass filter, as in the left column of Fig. 13. In the middle column, the attribute function is magnified and the resampling filter is dominated by the reconstruction kernel. Since the resampling filter unifies a reconstruction kernel and a low-pass filter, it provides a smooth transition between magnification and minification. Moreover, the reconstruction kernel is scaled anisotropically in situations where the volume is stretched in one direction and shrunken in the other, as shown in the right column. In the bottom row, we show the filter shapes resulting from uniformly scaling the reconstruction kernel to avoid aliasing, as proposed by Swan et al. [18]. Essentially, the reconstruction kernel is enlarged such that its minor radius is at least as long as the minor radius of the low-pass filter. For spherical reconstruction kernels, or circular footprint functions, this is basically equivalent to the EWA resampling filter. However, for elliptical footprint functions, uniform scaling leads to overly blurred images in the direction of the major axis of the ellipse.

We compare our method to Swan's method in Fig. 14. The images on the left were rendered with EWA volume

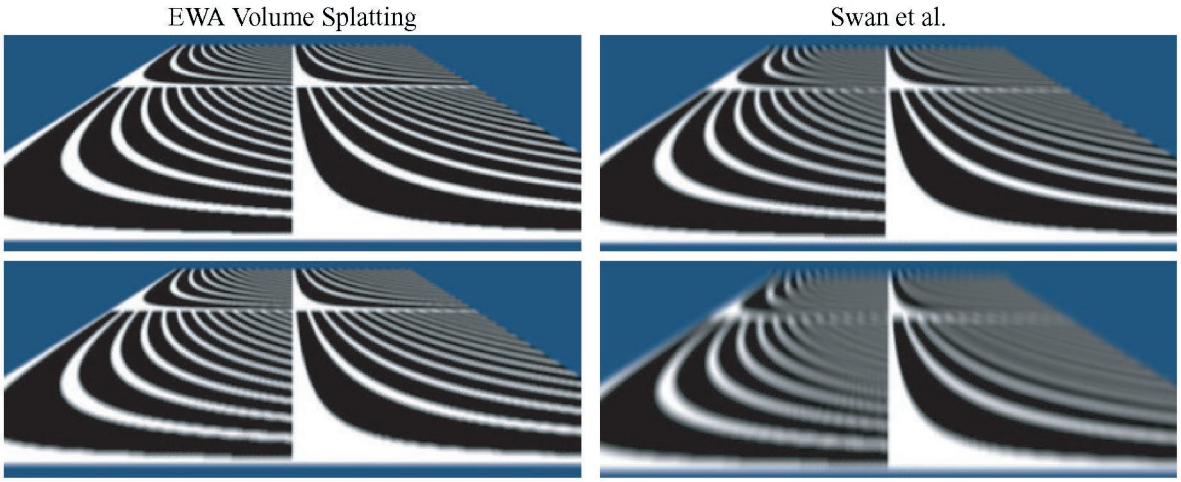


Fig. 14. Comparison between EWA volume splatting and Swan et al. Top two rows: $1,024 \times 512 \times 3$ volume texture. Bottom two rows: $1,024 \times 256 \times 3$ volume texture.

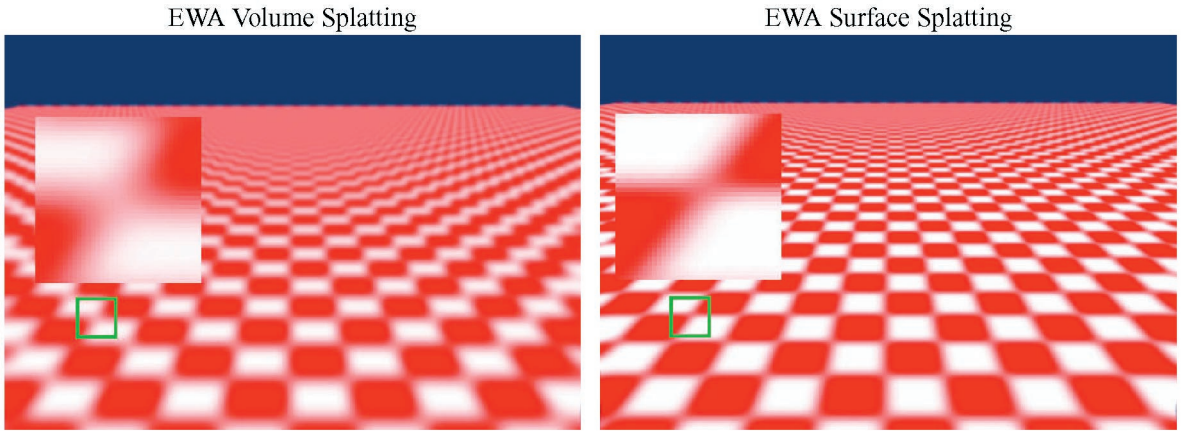


Fig. 15. EWA volume splatting versus EWA surface splatting; $512 \times 512 \times 3$ volume texture.

splats, those on the right with Swan's uniformly scaled kernels. We used rectangular zebra textures with x and y dimensions of $1,024 \times 512$ (in the first row) and $1,024 \times 256$ (in the second row) and mapped the textures to a square. This leads to elliptical reconstruction kernels with a ratio between the length of the major and minor radii of 2 to 1 and 4 to 1, respectively. Clearly, the EWA filter produces a crisper image and, at the same time, does not exhibit aliasing artifacts. As the ratio between the major and minor radii of the reconstruction kernels increases, the difference from Swan's method becomes more pronounced. For strongly anisotropic kernels, Swan's uniform scaling produces excessively blurred images, as shown on the right in Fig. 14. Each frame took approximately 6 seconds to render on an 866 MHz PIII processor.

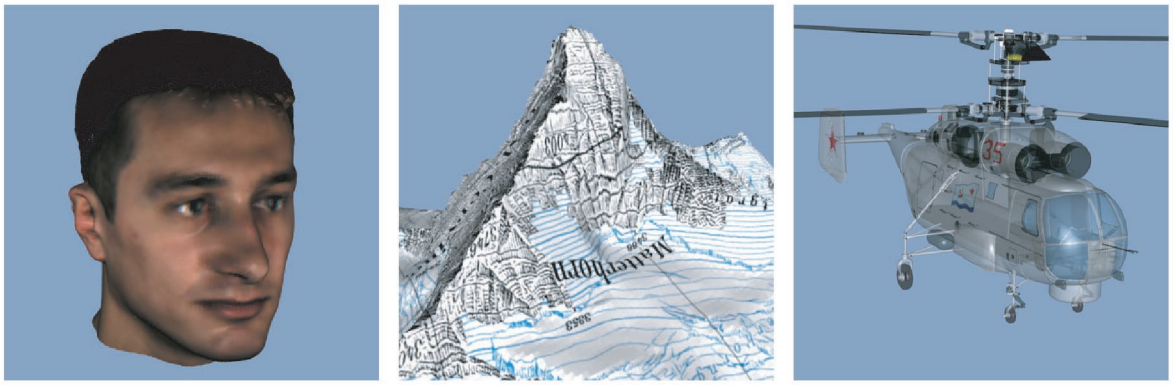
In Fig. 15, we compare EWA splatting using volume kernels on the left to surface reconstruction kernels on the right. The texture size is 512×512 in the x and y direction. Typically, the perspective projection of a spherical kernel is almost a circle. Therefore, rendering with volume kernels does not exhibit anisotropic texture filtering and produces textures that are slightly too blurry, similar to isotropic texture filters such as trilinear mipmapping. On the other hand, splatting surface kernels is equivalent to EWA texture filtering. Circular surface kernels are mapped to ellipses, which results in high image quality because of anisotropic filtering.

In Fig. 16, we show a series of volume renderings of the UNC CT scan of a human head ($256 \times 256 \times 225$), the UNC engine ($256 \times 256 \times 110$), and the foot of the visible woman data set ($152 \times 261 \times 220$). The texture in the last example is rendered using EWA surface splatting, too. The images illustrate that our algorithm correctly renders semitransparent objects as well. The skull of the UNC head, the bone of the foot, and the iso-surface of the engine were rendered with flattened surface splats oriented perpendicular to the volume gradient. All other voxels were rendered with EWA volume splats. Each frame took approximately 11 seconds to render on an 866 MHz PIII processor.

Fig. 17 shows results of EWA surface splatting which were rendered using a dedicated surface splatting renderer [23]. The face in Fig. 17a was acquired by a laser range scanner. Fig. 17b illustrates high quality texturing on terrain data and Fig. 17c shows semi-transparent surfaces on the complex model of a helicopter. Table 1 shows the performance of our unoptimized software implementation of EWA surface splatting. The frame rates were measured on a 1.1 GHz AMD Athlon system with 1.5 GByte memory. We rendered to a frame buffer with a resolution of 256×256 and 512×512 pixels, respectively.

(a) (b) (c)

Fig. 16. Semitransparent objects rendered using EWA volume splatting. The skull of the UNC head, the iso-surface of the engine, and the bone of the foot are rendered with flattened surface splats. (a) UNC head. (b) UNC engine. (c) Visible Woman foot.



(a) (b) (c)

Fig. 17. EWA surface splatting of a scan of a human face, textured terrain, and a complex point-sampled object with semi-transparent surfaces. (a) Scanned head. (b) Textured Terrain Data. (c) Semi-transparent surfaces.

9 Conclusions

We present a new splat primitive, called the EWA resampling filter. Using a general signal processing framework, we derive a formulation of the EWA resampling filter for both volume and surface splatting. Our primitive provides high quality antialiasing, combining an elliptical Gaussian reconstruction kernel with a Gaussian low-pass filter. We use a novel approach of computing the footprint function for volume rendering. Exploiting the mathematical features of 2D and 3D Gaussians, our framework efficiently handles arbitrary elliptical reconstruction kernels and perspective projection. Therefore, our primitive is suitable to render regular, rectilinear, curvilinear, and irregular volume data sets. Our formulation of the EWA surface resampling filter is equivalent to Heckbert's EWA texture filter. It provides high quality, anisotropic texture filtering for point-sampled surfaces. Hence, we call our primitive *universal*, facilitating the rendering of surface and volume data.

We have not yet investigated whether other kernels besides elliptical Gaussians may be used with this framework. In principle, a resampling filter could be derived from any function that allows the analytic evaluation of the operations described in Section 6.1 and that is a good approximation of an ideal low-pass filter.

To achieve interactive frame rates, we are currently investigating the use of graphics hardware to rasterize EWA splats as texture mapped polygons. Programmable vertex shaders of modern GPUs (graphics processing units) provide all operations to compute EWA resampling filters completely in hardware. To render nonrectilinear data sets, we are investigating fast back-to-front sorting algorithms. Furthermore, we want to experiment with our splat primitive in a postshaded volume rendering pipeline. The derivative of the EWA resampling filter could be used as a band-limited gradient kernel, hence avoiding aliasing caused by shading for noisy volume data.

TABLE 1
Rendering Performance for Fame Buffer Resolutions 256×256 and 512×512

Data	# Points	256×256	512×512
Scanned Head	429075	1.3 fps	0.7 fps
Matterhorn	4782011	0.2 fps	0.1 fps
Helicopter	987552	0.6 fps	0.3 fps

ACKNOWLEDGMENTS

The authors would like to thank Paul Heckbert for his encouragement and helpful comments and Ron Perry and Liu Ren for many stimulating discussions. Many thanks to Lisa Sobierajski Avila for her help with our implementation of EWA volume splatting in vtk. Thanks to Jennifer Roderick Pfister and Martin Roth for proofreading the paper.

REFERENCES

[1] L. Westover, "Interactive Volume Rendering," *Proc. Chapel Hill Workshop Volume Visualization,* C. Upson, ed., pp. 9-16, May 1989.
[2] L. Westover, "Footprint Evaluation for Volume Rendering," *Computer Graphics, Proc. SIGGRAPH '90,* pp. 367-376, Aug. 1990.
[3] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, J. Shade, and D. Fulk, "The Digital Michelangelo Project: 3D Scanning of Large Statues," *Computer Graphics, SIGGRAPH 2000 Proc.,* pp. 131-144, July 2000.
[4] H. Hoppe, T. DeRose, T. Duchampt, J. McDonald, and W. Stuetzle, "Surface Reconstruction from Unorganized Points," *Computer Graphics, SIGGRAPH '92 Proc.,* pp. 71-78, July 1992.
[5] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Computer Graphics, SIGGRAPH '96 Proc.,* pp. 303-312, Aug. 1996.
[6] S. Rusinkiewicz and M. Levoy, "Qsplat: A Multiresolution Point Rendering System for Large Meshes," *Computer Graphics, SIGGRAPH 2000 Proc.,* pp. 343-352, July 2000.
[7] J.P. Grossman and W. Dally, "Point Sample Rendering," *Rendering Techniques '98,* pp. 181-192, July 1998.
[8] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives," *Computer Graphics, SIGGRAPH 2000 Proc.,* pp. 335-342, July 2000.
[9] P. Heckbert, "Fundamentals of Texture Mapping and Image Warping," MS thesis, Dept. of Electrical Eng. and Computer Science, Univ. of California at Berkeley, June 1989.
[10] N. Greene and P. Heckbert, "Creating Raster Omnimax Images from Multiple Perspective views Using the Elliptical Weighted Average Filter," *IEEE Computer Graphics and Applications,* vol. 6, no. 3, pp. 21-27, June 1986.
[11] K. Mueller and R. Crawfis, "Eliminating Popping Artifacts in Sheet Buffer-Based Splatting," *Proc. IEEE Visualization '98,* pp. 239-246, Oct. 1998.
[12] A. Van Gelder and K. Kim, "Direct Volume Rendering with Shading via Three-Dimensional Textures," *Proc. ACM/IEEE Symp. Volume Visualization,* pp. 23-30, Oct. 1996.
[13] B. Cabral, N. Cam, and J. Foran, "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware," *Proc. 1994 Workshop Volume Visualization,* pp. 91-98, Oct. 1994.
[14] K. Mueller and R. Yagel, "Fast Perspective Volume Rendering with Splatting by Utilizing a Ray-Driven Approach," *Proc. IEEE Visualization '96,* pp. 65-72, Oct. 1996.
[15] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics, SIGGRAPH '91 Proc.,* pp. 285-288, July-Aug. 1991.
[16] L. Lippert and M.H. Gross, "Fast Wavelet Based Volume Rendering by Accumulation of Transparent Texture Maps," *Computer Graphics Forum,* vol. 14, no. 3, pp. 431-444. Aug. 1995.
[17] X. Mao, "Splatting of Non Rectilinear Volumes through Stochastic Resampling," *IEEE Trans. Visualization and Computer Graphics,* vol. 2, no. 2, pp. 156-170. June 1996.
[18] J.E. Swan, K. Mueller, T. Möller, N. Shareef, R. Crawfis, and R. Yagel, "An Anti-Aliasing Technique for Splatting," *Proc. 1997 IEEE Visualization Conf.,* pp. 197-204, Oct. 1997.
[19] K. Mueller, T. Moeller, J.E. Swan, R. Crawfis, N. Shareef, and R. Yagel, "Splatting Errors and Antialiasing," *IEEE Trans. Visualization and Computer Graphics,* vol. 4, no. 2, pp. 178-191, Apr.-June 1998.
[20] M. Levoy and T. Whitted, "The Use of Points as Display Primitives," Technical Report TR 85-022, Dept. of Computer Science, Univ. of North Carolina at Chapel Hill, 1985.
[21] T.W. Mark, L. McMillan, and G. Bishop, "Post-Rendering 3D Warping," *Proc. 1997 Symp. Interactive 3D Graphics,* pp. 7-16, Apr. 1997.
[22] J. Shade, S.J. Gortler, L. He, and R. Szeliski, "Layered Depth Images," *Computer Graphics SIGGRAPH '98 Proc.,* pp. 231-242, July 1998.
[23] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Surface Splatting," *Computer Graphics, SIGGRAPH 2001 Proc.,* July 2001.
[24] M. Zwicker, H. Pfister, J. Van Baar, and M. Gross, "Ewa Volume Splatting," *Proc. IEEE Visualization 2001,* pp. 29-36, Oct. 2001.
[25] K. Mueller, T. Moeller, and R. Crawfis, "Splatting without the Blur," *Proc. 1999 IEEE Visualization Conf.,* pp. 363-370, Oct. 1999.
[26] C. Wittenbrink, T. Malzbender, and M. Goss, "Opacity-Weighted Color Interpolation for Volume Sampling," *Proc. IEEE Symp. Volume Visualization,* pp. 431-444, Oct. 1998.
[27] J.T. Kajiya and B.P. Von Herzen, "Ray Tracing Volume Densities," *Computer Graphics, Proc. SIGGRAPH '84,* vol. 18, no. 3, pp. 165-174. July 1984.
[28] N. Max, "Optical Models for Direct Volume Rendering," *IEEE Trans. Visualization and Computer Graphics,* vol. 1, no. 2, pp. 99-108, June 1995.
[29] P. Lacroute and M. Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transform," *Computer Graphics, Proc. SIGGRAPH '94,* pp. 451-457, July 1994.
[30] M. Levoy, "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications,* vol. 8, no. 3, pp. 29-37, May 1988.
[31] X. Mao, L. Hong, and A. Kaufman, "Splatting of Curvilinear Volumes," *IEEE Visualization '95 Proc.,* pp. 61-68, Oct. 1995.
[32] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit,* second ed. Prentice Hall, 1998.
[33] N. Jouppi and C. Chang, "$\mathbf{z}^3$: An Economical Hardware Technique for High-Quality Antialiasing and Transparency," *Proc. Eurographics/SIGGRAPH Workshop Graphics Hardware 1999,* pp. 85-93, Aug. 1999.
[34] L. Carpenter, "The A-Buffer, an Antialiased Hidden Surface Method," *Computer Graphics, SIGGRAPH '84 Proc.,* vol. 18, pp. 103-108, July 1984.

Matthias Zwicker is in his last year of the PhD program at the Computer Graphics Lab at ETH Zürich, Switzerland. He has developed rendering algorithms and data structures for point-based surface representations. He has also extended this work toward high quality volume rendering. Other research interests concern compression of point-based data structures, acquisition of real world objects, and texturing of point-sampled surfaces.

Hanspeter Pfister received the PhD degree in computer science in 1996 from the State University of New York at Stony Brook. He received the MS degree in electrical engineering from the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, in 1991. He is associate director and a research scientist at MERL—Mitsubishi Electric Research Laboratories—in Cambridge, Massachusetts. He is the chief architect of VolumePro, Mitsubishi Electric's real-time volume rendering hardware for PCs. His research interests include computer graphics, scientific visualization, and computer architecture. His work spans a range of topics, including point-based rendering and modeling, 3D scanning and 3D photography, and computer graphics hardware. Dr. Pfister has taught courses at major graphics conferences, including SIGGRAPH, IEEE Visualization, and Eurographics. He is an associate editor of the *IEEE Transactions on Visualization and Computer Graphics* (*TVCG*), a member of the Executive Committee of the IEEE Technical Committee on Graphics and Visualization (TCVG), and has served as a member of international program committees of major graphics conferences. He is the general chair of the IEEE Visualization 2002 conference in Boston. He is a member of the ACM, ACM SIGGRAPH, IEEE, IEEE Computer Society, and Eurographics Association.

Jeroen van Baar received the MS degree in computer science from Delft University of Technology, The Netherlands, in 1998. He is working at MERL—Mitsubishi Electric Research Labsoratories—in Cambridge, Massachusetts, as a member of the technical staff. His areas of interest include the broad fields of computer graphics, scientific visualization, and computer vision.

Markus Gross received a degree in electrical and computer engineering and the PhD degree in computer graphics and image analysis, both from the University of Saarbrücken, Germany. He is a professor of computer science and the director of the Computer Graphics Laboratory of the Swiss Federal Institute of Technology (ETH) in Züurich. From 1990 to 1994, Dr. Gross was with the Computer Graphics Center in Darmstadt, where he established and directed the Visual Computing Group. His research interests include physics-based modeling, point-based methods, and multiresolution analysis. He has widely published and lectured on computer graphics and scientific visualization and he authored the book *Visual Computing* (Springer, 1994). Dr. Gross has taught courses at major graphics conferences including SIGGRAPH, IEEE Visualization, and Eurographics. He is an associate editor of *IEEE Computer Graphics and Applications* and has served as a member of international program committees of major graphics conferences. Dr. Gross was a papers cochair of the IEEE Visualization '99 and Eurographics 2000 conferences. He is a member of the IEEE, ACM, and of the Eurographics Association.

▷ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.